Available online at www.sciencedirect.com
ScienceDirect
Computers and Mathematics with Applications 55 (2008) 2395–2412
An International Journal
computers & mathematics
with applications
www.elsevier.com/locate/camwa

Homology algorithm based on acyclic subspace

Marian Mrozek[a,*], Paweł Pilarczyk[a,b,1], Natalia Żelazna[a]

[a] *Institute of Computer Science, Jagiellonian University, ul. Nawojki 11, 30-072 Kraków, Poland*
[b] *Georgia Institute of Technology, Atlanta, GA 30332-0160, USA*

Received 9 January 2007; received in revised form 14 May 2007; accepted 30 August 2007

Abstract

We present a new reduction algorithm for the efficient computation of the homology of a cubical set. The algorithm is based on constructing a possibly large acyclic subspace, and then computing the relative homology instead of the plain homology. We show that the construction of acyclic subspace may be performed in linear time. This significantly reduces the amount of data that needs to be processed in the algebraic way, and in practice it proves itself to be significantly more efficient than other available cubical homology algorithms.

Keywords: Homology algorithm; Cubical set; Cubical homology; Acyclic subspace; Smith diagonalization

1. Introduction

In this paper we introduce a new method for the computation of the homology of a cubical set X. The method is based on the elementary observation that

$$H_n(X) \cong \begin{cases} H_n(X, A) & \text{for } n \geq 1 \\ \mathbb{Z} \oplus H_n(X, A) & \text{for } n = 0 \end{cases}$$

if A is an acyclic subset of X. By an acyclic set we mean a set whose homology is the same as the homology of the space consisting of just one point. The key part of the presented method is the construction of a possibly large acyclic cubical subset A of X in an efficient way. Then we compute the relative homology of the pair (X, A) in order to obtain $H_*(X)$. Due to the excision property of the relative homology the cost of this computation depends only on the amount of cubes in X which are not in A, so it is relatively small when A is large.

We present our method for cubical homology, but it is straightforward to extend the method to simplicial homology.

1.1. Motivation

Among the first applications of the homology algorithm of triangulations were problems in computer-aided design [1]. Although in the classical homology theory simplicial complexes constitute the traditional combinatorial

* Corresponding author.
E-mail addresses: Marian.Mrozek@ii.uj.edu.pl (M. Mrozek), Pawel.Pilarczyk@ii.uj.edu.pl (P. Pilarczyk), zelazna@ii.uj.edu.pl (N. Żelazna).
[1] Current address: Kyoto University, Department of Mathematics, Kyoto 606-8502, Japan.

0898-1221/$ - see front matter
doi:10.1016/j.camwa.2007.08.044

representation of sets, in many present day applications cubical structure is more natural and convenient. For instance, in digital imaging the data is usually acquired and stored as a bitmap of pixels or voxels. In rigorous numerics of dynamical systems computations are performed by means of interval arithmetic [2], which also leads in a natural way to cubical sets. The need to compute the homology of a cubical set appeared probably for the first time in the computer assisted proof of chaos in the Lorenz equations [3,4]. In that case, just in lack of a better method, the homology was found just by a visual inspection. This was possible, because the set was planar and although it was large, its topology was very simple. However, this paper indicated the need for efficient cubical homology algorithms and in 1999 the first working implementations [5,6] of such algorithms appeared. This, together with new algorithms translating the problems in dynamics to problems in topology of cubical sets [7–10], enabled more advanced computer assisted proofs in dynamics [11–14]. On the other hand, the availability of cubical homology software enabled direct applications to image processing and recognition [15–18]. Other areas of applications of computational homology are: pattern classification [19,20], sensor networks [21], and materials science [22]. In many of these applications the number of cubes or simplices is counted in hundreds of thousands or even millions. All this creates the demand for faster homology algorithms.

1.2. Prior work

The classical homology algorithms reduce the problem to Smith diagonalization [23, Section 1.11]. The best available Smith diagonalization algorithms have supercubical complexity [24]. This is in general not sufficient when the number of building blocks of the topological space (cubes, simplices) is counted in thousands or more. The problem of efficient computation of homology groups has been addressed by many authors and from various points of view [1,25–32].

An alternative to various improvements of the Smith Normal Form algorithm, including probabilistic algorithms, are the methods of reduction originally proposed in [26] and then developed in [27,31,11,33]. The reduction methods consist in iterating the process of replacing the chain complex or even the topological space by a smaller one with the same homology and computing the homology only when no more reductions are possible. This way one postpones the process of computing the homology of the chain complex until the complex is small. Moreover, if the reduction process is applied directly to the topological space, then also the expensive process of constructing the chain complex is postponed until the space is small. Of course, one can profit from the reduction process only if one step of the reduction is computationally inexpensive and the reduction is significant. The present work may be characterized as an essentially new, fast and deep method of reducing the topological space.

We will use three earlier implementations of reduction methods as a reference point for the presented new method. (All these implementations are available from the webpage of the Computational Homology Project [34].)

PP Algebraic elementary reductions by P. Pilarczyk [5], based on [26].
BK Geometrically controlled algebraic reductions by W. Kalies [6], based on [27].
AR Algebraic elementary reductions by M. Mrozek [35], based on [26].

In the case of a cubical set the matrices of boundary maps are sparse. Unfortunately this is not very helpful, because of the fill-in process in the Smith algorithm. Nevertheless, it is reasonable to use this fact and all three implementations **PP**, **BK**, **AR** use an appropriate technique (respectively hashing tables, trees and lists) to avoid unnecessary manipulation of zeros.

1.3. Outline

We begin with recalling the concepts of cubical sets and cubical homology in Section 2. Then, in Section 3, an algorithm for the construction of an acyclic subspace is presented. In Section 4 we show how to use this algorithm to construct an efficient homology algorithm. Various algorithms for the acyclicity test used in Section 3 are discussed in Section 5. In Section 6 we show how to efficiently use the acyclic subspace homology algorithm in the presence of more than one connected component. In Section 7 we discuss a related method of preprocessing cubical sets, called by us *shaving*, which in some situations improves the speed of homology computations. Then, in Section 8, we present the implementation of the method for cubical sets. This implementation is compared with earlier implementations in Section 9, where some numerical experiments and benchmarks are presented. In Section 10 we compare our software with some other homology packages, not available from [34]. In the last section we present conclusions.

2. Preliminaries

In the present paper we assume that the reader is familiar with the concepts of homology theory, in particular the homology theory of cubical sets, as presented in [31]. However, to facilitate the understanding of the main results, we recall the basic notation and basic concepts.

2.1. Cubical sets

Throughout the paper the sets of natural numbers, integers, rational numbers and real numbers are denoted respectively by $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$ and $\mathbb{R}$. For a finite set Z its cardinality is denoted by $\operatorname{card} Z$. Given sets $A \subset X \subset \mathbb{R}^d$ we denote by $\operatorname{int}_X A$, the interior of A in X.

An *elementary interval* in $\mathbb{R}$ is an interval of the form $[k, k+\delta]$, where $k \in \mathbb{Z}$ and $\delta \in \{0, 1\}$. If $\delta = 0$, the interval is called *degenerate*. Let $d \in \mathbb{N}$ be fixed. The Cartesian product of d elementary intervals is called an *elementary cube* in $\mathbb{R}^d$. The dimension of the elementary cube Q is the number of elementary intervals in the product which are not degenerate. The family of all q-dimensional elementary cubes in $\mathbb{R}^d$ is denoted by $\mathcal{K}_q^d$ and the family of all elementary cubes in $\mathbb{R}^d$ is denoted by $\mathcal{K}^d$. The elements of $\mathcal{K}_0^d$ and $\mathcal{K}_1^d$ are referred to respectively as *vertices* and *edges*.

A subset $X \subset \mathbb{R}^d$ is called a *cubical set* if it is a finite union of elementary cubes. A finite subfamily of $\mathcal{K}^d$ is called a *cubical family*.

2.2. Cubical homology

A q-chain is a function $c : \mathcal{K}_q^d \to \mathbb{Z}$ which vanishes at all but a finite number of cubes. The *support* of the chain c is given by

$$|c| := \bigcup \{Q \in \mathcal{K}_q^d \mid c(Q) \neq 0\}.$$

Given a cubical set X put

$$C_q(X) := \{c \in C_q \mid |c| \subset X\}.$$

Then $C_q(X)$ with the argumentwise addition is a free abelian group. The set of all elementary chains of the form

$$\widehat{Q}(P) = \begin{cases} 1 & \text{if } P = Q \\ 0 & \text{otherwise} \end{cases}$$

for all $Q \in \mathcal{K}_q^d$ such that $Q \subset X$, is a basis of $C_q(X)$. Given two elementary cubes $P \in \mathcal{K}_{q_1}^d$ and $Q \in \mathcal{K}_{q_2}^d$, we define the *cubical product* of the chains $\widehat{P}$, $\widehat{Q}$, by

$$\widehat{P} \diamond \widehat{Q} := \widehat{P \times Q}$$

and we extend this definition linearly to arbitrary chains.

We define the *boundary operator* as the homeomorphism $\partial : C_q \to C_{q-1}$ given recursively on generators by

$$\partial \widehat{Q} := \begin{cases} 0 & \text{if } Q = [l], \\ \widehat{[l+1]} - \widehat{[l]} & \text{if } Q = [l, l+1]. \\ \partial \widehat{I} \diamond \widehat{P} + (-1)^{\dim I} \widehat{I} \diamond \partial \widehat{P} & \text{if } Q = I \times P \\ & \text{for } I \in \mathcal{K}^1 \text{ and } P \in \mathcal{K}^{d-1}. \end{cases}$$

One can verify that for every cubical set X we have an induced boundary operator $\partial_q^X : C_q(X) \to C_{q-1}(X)$ and $\partial_q^X \partial_{q+1}^X = 0$. Therefore, $B_q(X) := \operatorname{im} \partial_{q+1}^X$, the image of ∂_{q+1}^X is a subgroup of $Z_q(X) := \ker \partial_q^X$, the kernel of ∂_q^X. The quotient group

$$H_q(X) := Z_q(X)/B_q(X)$$

is called the qth *cubical homology group* of X. By the *homology* of X, we mean the collection of all homology groups $H(X) := \{H_q(X)\}$.

2.3. Full cubical sets

A d-dimensional elementary cube in $\mathbb{R}^d$ is called a *full* elementary cube. The family of all full elementary cubes will be denoted by $\mathcal{H}^d$ or simply by $\mathcal{H}$. A special case of a cubical set is the *full cubical set*, i.e. a finite union of full elementary cubes. Similarly, a *full cubical family* is a cubical family consisting of full cubical sets. For every full cubical set X, there exists a unique full cubical family $\mathcal{X} \subset \mathcal{H}$ such that

$$|\mathcal{X}| := \bigcup \mathcal{X} = X.$$

Then $\mathcal{X}$ is called the *representation* of X and X is called the *geometric realization* of $\mathcal{X}$. In what follows we emphasize the formal difference between the cubical sets and cubical families by denoting the latter with calligraphic letters. However, we often freely carry over the terminology from cubical sets to their representations as families of cubes. For instance, by the homology of $\mathcal{X} \subset \mathcal{H}$ we understand the homology of $|\mathcal{X}|$ and we say that $\mathcal{X}$ is acyclic meaning that $|\mathcal{X}|$ is acyclic.

A *d-dimensional bitmap* is a d-dimensional array of bits which correspond to pixels, voxels, or higher-dimensional "boxes" which we will simply call *pixels*, independently of their dimension. A bitmap represents a cubical set in the following way: The pixel with the coordinates $(k_1, \ldots, k_d)$ corresponds to the cube $[k_1, k_1 + 1] \times \cdots \times [k_d, k_d + 1]$, and the set represented by the bitmap is the union of all the cubes whose corresponding pixels are set to 1.

A full elementary cube P is said to be a *neighbor* of a full elementary cube Q if $P \cap Q \neq \emptyset$. In terms of the corresponding pixels, this means that all their coordinates differ by no more than 1. The full elementary cube P is called a *neighbor* of a full cubical family $\mathcal{X} \subset \mathcal{H}$ if P is a neighbor of at least one cube $Q \in \mathcal{X}$. A *neighborhood* of Q is the set

$$o(Q) := \{P \in \mathcal{H} : P \cap Q \neq \emptyset\}$$

and for a full cubical family $\mathcal{X}$ we define $o_{\mathcal{X}}(Q) := \mathcal{X} \cap o(Q)$.

Because of the way the geometric realization of a cubical family is defined, two cubes that intersect along lower-dimensional faces, such as vertices or edges, are treated as adjacent. In the imaging literature, this is often referred to as considering $(3^d - 1)$-neighborhoods rather than $2d$-neighborhoods, where d is the dimension of the underlying space.

Since in the sequel we consider only full elementary cubes, we drop the words *full* and *elementary* when referring to full elementary cubes. We also drop the word *full*, when referring to full cubical sets and full cubical families.

3. Acyclic subspace construction

In this section we present the algorithm for the construction of an acyclic cubical subset of a given cubical set.

The idea of the algorithm is to begin with a set $\mathcal{A}$ that contains a single cube selected arbitrarily from $\mathcal{X}$ and extending this set by gradually adding cubes $Q \in \mathcal{X} \setminus \mathcal{A}$ such that $|\mathcal{A}| \cup Q$ is acyclic. This may be considered as a variant of Algorithm 4.5 in [33]. We use a queue to store the neighbors of cubes added to $\mathcal{A}$, as these are potential candidates for addition to $\mathcal{A}$ in the next step.

All our tests for the acyclicity of $|\mathcal{A}| \cup Q$ presented in Section 5 are based on the following elementary observation.

Lemma 1. *If $\mathcal{A}$ is an acyclic cubical family and $Q \in \mathcal{H} \setminus \mathcal{A}$, then $|\mathcal{A}| \cup Q$ is acyclic if and only if $|\mathcal{A} \cap o(Q)|$ is acyclic.*

Proof. Put $\mathcal{A}' := \mathcal{A} \cap o(Q)$. By [11, Lemma 9] the set $|\mathcal{A}'| \cap Q$ is a deformation retract of $|\mathcal{A}'|$. Since obviously $|\mathcal{A}| \cap Q = |\mathcal{A}'| \cap Q$, we conclude that $|\mathcal{A}| \cap Q$ is a deformation retract of $|\mathcal{A} \cap o(Q)|$. Therefore, by [31, Theorems 6.65 and 6.69], the set $|\mathcal{A}| \cap Q$ is acyclic if and only if $|\mathcal{A} \cap o(Q)|$ is acyclic and it remains to be proved that $|\mathcal{A}| \cup Q$ is acyclic if and only if $|\mathcal{A}| \cap Q$ is acyclic.

Since $|\mathcal{A}|$ is acyclic by assumption and Q is acyclic by [31, Theorem 2.76], the acyclicity of $|\mathcal{A}| \cup Q$ follows from the acyclicity of $|\mathcal{A}| \cap Q$ by [31, Theorem 2.78]. Now assume that $|\mathcal{A}| \cup Q$ is acyclic. It follows from Mayer–Vietoris Theorem (see [31, Theorem 9.29]) that

$$H_k(|\mathcal{A}| \cap Q) \cong H_{k+1}(|\mathcal{A}| \cup Q) = 0 \quad \text{for } k \in \mathbb{N}.$$

Thus we need to prove only that $H_0(|\mathcal{A}| \cap Q) \cong \mathbb{Z}$. Let V_1, V_2 be two vertices in $|\mathcal{A}| \cap Q$. Since $|\mathcal{A}|$ and Q are acyclic, there exists chains $c_1 \in C_1(|\mathcal{A}|)$ and $c_2 \in C_1(Q)$ such that $\partial c_1 = \widehat{V}_1 - \widehat{V}_2 = \partial c_2$. Then $z := c_1 - c_2 \in Z_1(|\mathcal{A}| \cup Q)$.

Since $|\mathcal{A}| \cup Q$ is acyclic, there exists a $d \in C_2(|\mathcal{A}| \cup Q)$ such that $\partial d = z$. By [31, Proposition 2.77] there exist $d_1 \in C_2(|\mathcal{A}|)$ and $d_2 \in C_2(Q)$ such that $d = d_1 + d_2$. Then

$$c_1 - c_2 = z = \partial d_1 + \partial d_2$$

and

$$u := c_1 - \partial d_1 = c_2 + \partial d_2 \in C_1(|\mathcal{A}| \cap Q).$$

Therefore, $\partial u = \partial c_1 = \widehat{V}_1 - \widehat{V}_2$, which implies that $\widehat{V}_1$ and $\widehat{V}_2$ are homologous in $|\mathcal{A}| \cap Q$ and consequently $H_0(|\mathcal{A}| \cap Q) \cong \mathbb{Z}$. □

In the following formal description of the acyclic subspace algorithm, it is assumed that the function `AcyclicityTest` $(\mathcal{A}, Q)$ is *admissible* in the sense that it either returns true meaning that $\mathcal{A} \cap o(Q)$ is acyclic or it returns false meaning that it failed to prove that $\mathcal{A} \cap o(Q)$ is acyclic.

Several possible choices for this function are described in detail in Section 5.

Algorithm 2. Acyclic Subspace

```
function AcyclicSubspace (cubical family 𝒳)
begin
  Q := any cube from 𝒳;
  𝒜 := {Q};
  𝒬 := empty queue of cubes;
  for each P ∈ 𝒳 \ {Q} ∩ o(Q) do
    enqueue (𝒬, P);
  while 𝒬 ≠ ∅ do begin
    Q := dequeue (𝒬);
    if AcyclicityTest(𝒜, Q) then begin
      𝒜 := 𝒜 ∪ {Q};
      for each P ∈ (𝒳 \ 𝒜) ∩ o(Q) do
        if P ∉ 𝒬 then
          enqueue(𝒬, P);
    end;
  end;
  return 𝒜;
end;
```

Note that in the actual implementation one might use two parallel data structures to represent $\mathcal{Q}$: a set and a queue, because in addition to the standard queue operations, in the algorithm we also need to verify whether a given cube is already contained in the queue or not, and this operation is normally not very efficient for queues.

Every time a cube is added to the constructed acyclic subset, all its neighbors from $\mathcal{X} \setminus \mathcal{A}$ are enqueued. This construction has two advantages. First, only the cubes whose addition to $\mathcal{A}$ preserves its connectedness are considered in the next step, which prevents from analysing cubes disjoint from $|\mathcal{A}|$. Second, the oldest neighbors in the queue are processed first, which helps grow the set $\mathcal{A}$ in a balanced way.

Theorem 3. *Assume that the function* `AcyclicityTest` *in Algorithm 2 is admissible. Then Algorithm 2 called with a nonempty cubical family $\mathcal{X} \subset \mathcal{H}$ returns an acyclic subset $\mathcal{A}$ of $\mathcal{X}$. Moreover, the "***while***" loop is passed at most $(3^d - 1)\operatorname{card}\mathcal{X}$ times.*

Proof. Put

$$K := \{k \in \mathbb{N} \mid \text{the "}\textbf{while}\text{" loop is passed at least } k \text{ times}\}.$$

For $k \in K$ let $\mathcal{A}_k$ and Q_k denote respectively the contents of variable $\mathcal{A}$ and Q on entering the kth pass of the "**while**" loop. Since the family $\mathcal{A}_1$ contains just one cube from $\mathcal{X}$, it is acyclic by [31, Theorem 2.76]. If $k - 1, k \in K$, then either $\mathcal{A}_k = \mathcal{A}_{k-1}$ or $\mathcal{A}_k = \mathcal{A}_{k-1} \cup \{Q_k\}$. The latter case only happens when the function `AcyclicityTest` called with $\mathcal{A}_{k-1}$ and Q_{k-1} returns **true**. The admissibility of the function `AcyclicityTest` and Lemma 1 imply that the

set $\mathcal{A}_k$ is acyclic in this case, too. By induction, all the sets $\mathcal{A}_k$ for $k \in K$ are acyclic. It remains to be shown that the "**while**" loop runs only at most $(3^d - 1)\mathrm{card}\,\mathcal{X}$ times. Let $q_0 := 0$ and for $k \in K$ let q_k and a_k denote respectively the number of elements in the queue $\mathcal{Q}$ and the number of elements in the set $\mathcal{A}$ on entering the kth pass of the "**while**" loop. Note that $a_{k+1} - a_k \in \{0, 1\}$ whenever $k, k+1 \in K$. For $i \in \mathbb{N}$ put

$$k_i := \max\{l \in K \mid a_l \leq i\}.$$

Let $I := \{i \in \mathbb{N} \mid k_i < k_{i+1}\}$. Observe that $i \in I$ implies that the acyclicity test succeeds on the k_ith pass of the "**while**" loop, $a_{k_i} = i$ and $a_{k_i+1} = i + 1$. For $i \in I$, let s_i denote the number of cubes added to the queue $\mathcal{Q}$ inside the block following the acyclicity test on the k_ith pass of the "**while**" loop.

For $k \in K$ such that $k + 1 \in K$ define $r_k := q_{k+1} - q_k$. Observe that

$$r_k = \begin{cases} -1 + s_i & \text{if } k = k_i \text{ for some } i \in I, \\ -1 & \text{otherwise.} \end{cases}$$

Now we have for $k \in K$

$$q_k = \sum_{i=0}^{k} r_k = -k + \sum_{i=1}^{a_k} s_i \leq -k + a_k(3^d - 1) \leq -k + (3^d - 1)\mathrm{card}\,\mathcal{X}. \tag{1}$$

Therefore, there exists a $k_0 \in \mathbb{N}$ such that $q_{k_0} = 0$. This implies that the "**while**" loop terminates on the k_0th pass of the "**while**" loop. Moreover, it follows from (1) that $k_0 \leq (3^d - 1)\mathrm{card}\,\mathcal{X}$, which completes the proof. □

As an immediate corollary we get

Corollary 4. *Assume the space dimension d is fixed and the function* `AcyclicityTest` *in Algorithm 2 runs in constant time. Let n denote the cardinality of the cubical family on input of Algorithm 2. Then Algorithm 2 runs in $O(n)$ time when the cubical families are implemented as bit arrays (bitmaps) and in $O(n \log n)$ time when the cubical families are implemented as binary search trees.*

4. Acyclic subspace homology algorithm

Assume that `Homology`$(\mathcal{X}, \mathcal{A})$ is a function which, given on input cubical families $\mathcal{A} \subset \mathcal{X}$, returns the relative homology $H_*(|\mathcal{X}|, |\mathcal{A}|)$ (for examples of algorithms which may be used to implement such a function see [31,36]). As we explained in the introduction, the computation of the homology of $|\mathcal{X}|$ may be replaced by the computation of the relative homology of the pair $(|\mathcal{X}|, |\mathcal{A}|)$ for some subfamily $\mathcal{A}$ such that $|\mathcal{A}|$ is acyclic. As we show in the proof of Theorem 6, the computation of $H_*(|\mathcal{X}|, |\mathcal{A}|)$ may be replaced by the computation of $H_*(|\mathcal{X}_0|, |\mathcal{A}_0|)$, where $\mathcal{A}_0 := o(\mathcal{X} \setminus \mathcal{A}) \cap \mathcal{A}$ and $\mathcal{X}_0 := \mathcal{X} \setminus \mathcal{A} \cup \mathcal{A}_0$. This is where we profit from our approach, because when $\mathcal{X} \setminus \mathcal{A}$ is small, then also $\mathcal{X}_0$ and $\mathcal{A}_0$ are small compared to $\mathcal{X}$ and in consequence the homology of the pair $(|\mathcal{X}_0|, |\mathcal{A}_0|)$ may computed much faster than the homology of $|\mathcal{X}|$.

Therefore, the cubical homology algorithm based on acyclic subspace construction is as follows:

Algorithm 5. Acyclic Subspace Homology Algorithm

function `AS_Homology(cubical family` $\mathcal{X}$`)`
begin
  $\mathcal{A}$:= `AcyclicSubspace(`$\mathcal{X}$`)`;
  $\mathcal{Z} := \mathcal{X} \setminus \mathcal{A}$;
  $\mathcal{A}_0 := o(\mathcal{Z}) \cap \mathcal{A}$;
  $\mathcal{X}_0 := \mathcal{Z} \cup \mathcal{A}_0$;
return `Homology(`$\mathcal{X}_0, \mathcal{A}_0$`)`;
end;

Theorem 6. *Algorithm 5 called with a cubical family $\mathcal{X}$ on input returns the reduced homology of $|\mathcal{X}|$.*

Proof. Let X, A, X_0, A_0 denote the geometric realizations of $\mathcal{X}, \mathcal{A}, \mathcal{X}_0$ and $\mathcal{A}_0$ respectively. It follows from the acyclicity of A and from the exact sequence of the pair (X, A) (see [31, Corollary 9.26]) that the homology groups $H_n(X)$ and $H_n(X, A)$ are isomorphic for $n > 0$ and the sequence

$$0 \to \mathbb{Z} \to X_0(X) \to X_0(X, A) \to 0$$

is exact. Therefore, $H_0(X, A)$ is isomorphic to $H_0(X)/\mathbb{Z}$, i.e. it is isomorphic to the 0th reduced homology of X.

Now put $U := \mathrm{int}_X|\mathcal{A} \setminus \mathcal{A}_0|$. One can easily verify that $X_0 = X \setminus U$, $A_0 = A \setminus U$ and U is a representable set in the sense of [31, Definition 6.1]. Therefore, it follows from [31, Theorem 9.14] that the inclusion $\iota : (X_0, A_0) \to (X, A)$ induces an isomorphism between $H_*(X_0, A_0)$ and $H_*(X, A)$. □

5. Acyclicity tests

In this section we present some algorithms which may be used to implement the function `AcyclicityTest` $(\mathcal{A}, Q)$. Recall that we only require this function to be admissible. In other words, we assume that the test is a partial test: when the function returns true, then the family $\mathcal{A} \cup o(Q)$ is acyclic but when it returns false, it does not imply that $\mathcal{A} \cup o(Q)$ is not acyclic. Of course, a total test, when the function returns true if and only if the family $\mathcal{A} \cup o(Q)$ is acyclic, might seem to be a better choice, but as we will see in the sequel this is not true in general, because a total test may be computationally expensive.

5.1. Direct computation

The simplest and most straightforward choice for a total acyclicity test is to use an independent homology algorithm to make the verification if $\mathcal{A} \cap o(Q)$ is acyclic. Using a slow homology algorithm to construct a faster homology algorithm need not be a total nonsense, especially in low dimensions, because the slow homology algorithm will be applied only to $\mathcal{A} \cap o(Q)$, which is small in low dimensions. Unfortunately, as we will see in Section 9, numerical experiments do not indicate that one can profit from this approach.

5.2. Direct homology computation via tabulated configurations

In theory, one can index all the subsets (configurations) of $o(Q)$, compute their homology to determine their acyclicity, store this information in a table `AcyclicConfigTest` and use the table for the acyclicity test.

Algorithm 7. Tabulated Configurations

function `TabConfTest(cubical family` $\mathcal{A}$, `cube` Q`)`
begin
  $\mathcal{N} := o(Q) \cap \mathcal{A}$;
  n := `index of` $\mathcal{N}$;
  return `AcyclicConfigTest[`n`]`;
end;

The number of such configurations in d-dimensional space is $c_d := 2^{3^d-1}$. Starting from dimension 4, when $c_4 = 2^{80}$, the method is of no practical value but in dimensions 2 and 3 the method may be implemented, and it leads to an extremely fast version of Algorithm 5, as we will show in Section 9.

5.3. Simple intersection

Since every polyhedron is homeomorphic to a cubical subcomplex of the boundary of a cube (see [37, Part III, Theorem 1.1] or [31, Theorem 11.17]), a total acyclicity test would essentially have to contain a complete homology algorithm. Therefore, a well-chosen partial test might turn out to be a better choice for dimension where we cannot tabulate all the neighborhood configurations. Probably the simplest nontrivial partial test for the acyclicity of a family $\mathcal{A}$ consists in verifying whether the intersection of the family is nonempty.

Algorithm 8. Simple Intersection

function `SimpleIntersection(cubical family` $\mathcal{A}$, `cube` Q`)`
begin
  $\mathcal{N} := o(Q) \cap \mathcal{A}$;
  return $\bigcap \mathcal{N} \neq \emptyset$;
end;

Theorem 9. *Assume Algorithm 8 is called with a family $\mathcal{A} \subset \mathcal{H}$ and a cube $Q \in \mathcal{H} \setminus \mathcal{A}$. If it returns **true**, then $|\mathcal{A}| \cap Q$ is acyclic.*

Proof. Assume Algorithm 8 returns **true**. This implies that there exists an $x \in \bigcap \mathcal{N}$. Therefore, $|\mathcal{N}|$ is star-shaped in the sense of [31, Definition 2.82]. Thus it is acyclic by [31, Proposition 2.84]. □

The function `SimpleIntersection`($\mathcal{A}$, Q) is fast, but obviously it is very far from a total criterion. Nevertheless, as we will show in Section 9, even this simple function may lead to a significant improvement in the computation of the homology of cubical sets.

5.4. Recursive approach

Further improvement may be obtained by calling the function `AcyclicSubspace` implemented on the basis of some simple partial test for acyclicity to obtain a better test for acyclicity.

Algorithm 10. Recursive Test

```
function RecursiveTest (cubical family A, cube Q)
begin
  if AcyclicityTest(A, Q) then
    return true;
  else begin
    C := A ∩ o(Q);
    return C = AcyclicSubspace (C);
  end;
end;
```

Theorem 11. *Assume* `AcyclicityTest` *is admissible and Algorithm 10 is called with a family $\mathcal{A} \subset \mathcal{H}$ and a cube $Q \in \mathcal{H} \setminus \mathcal{A}$. If it returns* **true***, then $|\mathcal{A}| \cap Q$ is acyclic.*

Proof. The algorithm returns true either if `AcyclicityTest`($\mathcal{A}$, Q) succeeds or when `AcyclicSubspace` applied to $\mathcal{C} := \mathcal{A} \cap o(Q)$ returns $\mathcal{C}$. If `AcyclicityTest`($\mathcal{A}$, Q) succeeds, then $|\mathcal{A}| \cap Q$ is acyclic by the admissibility of `AcyclicityTest`. If `AcyclicSubspace` ($\mathcal{C}$) returns $\mathcal{C}$, then obviously $\mathcal{A} \cap o(Q) = \mathcal{C}$ is acyclic, too. □

6. Connected components

Obviously, an acyclic subset of a set X is contained in a connected component of X. Therefore, if X has more than one connected component, then it is reasonable to construct an acyclic subspace for every component separately, use the acyclic subspaces to find the homology of the components and then take the direct sum of the homology of the components to get the homology of X. However, to speed up the computations, one can combine the algorithm for the acyclic subspace with the algorithm for the connected components.

Algorithm 12. Connected Component

```
function ConnectedComponent (cubical family X, A)
begin
  Q := empty queue of cubes;
  for each Q ∈ A do
    enqueue (Q, Q);
  while Q ≠ ∅ do begin
    Q := dequeue (Q);
    A := A ∪ {Q};
    for each p ∈ (X \ A) ∩ o(Q) do
      if p ∉ Q then
        enqueue (Q, p);
  end;
  return A;
end;
```

Theorem 13. *Algorithm 12 called with a cubical family $\mathcal{X}$ and its nonempty subset $\mathcal{A}$ such that $|\mathcal{A}|$ is connected returns the family $\mathcal{B}$ such that $|\mathcal{B}|$ is a connected component of $|\mathcal{X}|$, and $\mathcal{A} \subset \mathcal{B}$. Moreover, the loop is passed at most* card $\mathcal{X}$ *times.*

Proof. The inclusion $\mathcal{A} \subset \mathcal{B}$ is obvious. Since each cube $Q \in \mathcal{X}$ is enqueued at most once into the queue Q, and on each pass of the "**while**" loop exactly one cube is dequeued from this queue, the number of passes of the "**while**" loop does not exceed the number of cubes in $\mathcal{X}$. Let n denote the total number of passes of the "**while**" loop.

We will now prove that $|\mathcal{B}|$ is a connected component of $|\mathcal{X}|$. By [31, Corollary 2.57] it is enough to prove that $|\mathcal{B}|$ is an edge connected component of $|\mathcal{X}|$. Let V be a vertex of $|\mathcal{A}|$ and let E denote the edge connected component of V in $|\mathcal{X}|$. For $i = 1, 2, \ldots, n$ let $\mathcal{A}_{i-1}$ denote the value of the variable $\mathcal{A}$ just before the ith pass of the "**while**" loop and let Q_i denote the value of the variable Q in the ith pass of the loop. Obviously, $\mathcal{A}_0 \subset \mathcal{A}_{i-1} \subset \mathcal{A}_i$ and $\mathcal{A}_i = \mathcal{A}_{i-1} \cup \{Q_i\}$ for each $i = 1, \ldots, n$. Suppose $|\mathcal{A}_{k-1}|$ is connected for some $k \in \mathbb{N}$. Since $|\mathcal{A}_k| = Q_k \cup |\mathcal{A}_{k-1}|$, to show the connectedness of $|\mathcal{A}_k|$ it is enough to prove that $Q_k \cap |\mathcal{A}_{k-1}| \neq \emptyset$. This is true, because Q_k, as an element of the queue $\mathcal{Q}$ was either enqueued as an element of $\mathcal{A}_0 \subset \mathcal{A}_{k-1}$ or as a neighbor of the cube Q_j for some $j < k$ and then $Q_k \cap |\mathcal{A}_{k-1}| \supset Q_k \cap Q_j \neq \emptyset$. Since the connectedness of $|\mathcal{A}_0| = |\mathcal{A}|$ is assumed in the theorem, by induction $|\mathcal{B}| = |\mathcal{A}_n|$ is connected. In particular $|\mathcal{B}| \subset E$.

Now assume that the opposite inclusion does not hold. Since E is a cubical set (see [31, Proposition 2.56 and Corollary 2.57]), we can choose a vertex W in $E \setminus |\mathcal{B}|$. Let $V_0, V_1, \ldots, V_m$ be an edge path in E such that $V_0 = V$, $V_m = W$ and let i be the last index such that the edge joining V_{i-1} and V_i is contained in $|\mathcal{B}|$. Then there exists a cube P such that $V_i \in P \setminus |\mathcal{B}|$ and an index k such that $V_i \in Q_k$. It follows that $P \in o(Q_k)$, which implies that P is enqueued to the queue Q on some pass of the "**while**" loop. This implies that $P \in |\mathcal{B}|$, a contradiction. □

After having constructed the whole connected component of X containing A, we remove this component and repeat this procedure for the remaining family until we obtain the empty set. Algorithm 5 combined with this idea is as follows:

Algorithm 14. Acyclic Subspace Homology and Conn. Components

```
function AS_Homology_C2 (cubical family 𝒳)
begin
  result := 0;
  while 𝒳 ≠ ∅ do begin
    𝒜 := AcyclicSubset(𝒳); (Algorithm 2)
    ℬ := ConnectedComponent(𝒳, 𝒜); (Algorithm 12)
    result := result ⊕H_*(ℬ, 𝒜);
    𝒳 := 𝒳 \ ℬ;
  end;
  return result;
end;
```

7. Shaving

We will say that $Q \in \mathcal{X}$ is *removable* from $\mathcal{X}$ if $Q \cap |\mathcal{X} \setminus \{Q\}|$ is acyclic.

Proposition 15 (*See [33, Lemma 7.1]*). *If $Q \in \mathcal{X} \subset \mathcal{H}$ is removable from $\mathcal{X}$, then the homology of $\mathcal{X} \setminus \{Q\}$ is isomorphic to the homology of $\mathcal{X}$.*

Therefore, cubes removable from $\mathcal{X}$ may be removed from $\mathcal{X}$ without changing the homology of $\mathcal{X}$. Of course, after removing a removable cube Q from $\mathcal{X}$, the removability status of the remaining cubes may change, so the removability condition of the remaining cubes needs to be checked with respect to $\mathcal{X} \setminus \{Q\}$. By iterating the procedure of removing the removable cubes we obtain the process which we will call *shaving*. Shaving is not a new idea (see [11, Algorithm 10], for example). Gameiro and Nanda [38] modified Kalies' homology software **BK** [6] by preprocessing the homology computations with shaving. The modified software was used in [39]. A variant of shaving in the case of

relative homology was used in [33].

The advantage of shaving is the fact that the removable cubes may be searched in an arbitrary order, so there is no need to keep a queue of neighbors as in the case of constructing an acyclic subspace. The simplest reduction algorithm based on removing the removable cubes is as follows.

Algorithm 16. Shave

```
function Shave (var cubical family 𝒳)
begin
  for each Q ∈ 𝒳 do
    if AcyclicityTest(𝒳 \ {Q}, Q) then
      𝒳 := 𝒳 \ {Q};
end;
```

It follows immediately from Proposition 15 that the homology of $\mathcal{X}$ remains constant in course of running Algorithm 16. In many situations running this algorithm as a preprocessor to any other homology algorithm is a good idea, because the algorithm is very fast and may substantially reduce the original cubical family, speeding up the homology computations. Obviously, Algorithm 16 does not guarantee that the resulting cubical family does not admit a removable cube anymore, so one might run this algorithm a few times. The question how many times crucially depends on the particular cubical family. Actually, if the family contains very few removable cubes, preprocessing homology computations with any kind of shaving may slow down the computations. We will compare various versions of the acyclic subspace construction with and without shaving in Section 9.

8. Implementation

The acyclic subspace cubical homology algorithms based on the four variants of the acyclicity test described in Section 5 have been implemented by the first author. These implementations are

ASH – based on direct homology computations (Section 5.1)
ASLT – based on lookup tables (Section 5.2)
AS – based on simple intersection (Section 5.3)
ASR – based on recursive approach (Section 5.4).

All four implementations use the **AR** implementation of [26] after the acyclic subspace is constructed.

The implementations together with the benchmark programs used to prepare the timings presented in Section 9 are available at [35]. These implementations also constitute a part of the Computer Assisted Proofs in Dynamics (CAPD) software library [40] and Computational Homology Project (CHOMP) software library [34].

The implementations are written in C++ using the techniques of templates and generic programming to ensure both the high efficiency (to guarantee the good performance) and the high level abstraction (to ensure the reusability of the general code in various settings). For the moment, the software is available for cubical sets implemented as bitmaps. Bitmaps provide a memory-efficient and access-time-efficient method of storing cubical sets. For example, a three-dimensional rectangular area of the size $256 \times 256 \times 256$ filled with cubes in 50%, i.e. containing about 8 million cubes as in the case of cubical sets described in Section 9.4 takes up merely 2 MB of memory. The same amount of memory used to store a cubical set as a list of triples of one byte coordinates would allow for only 2/3 of a million cubes and the access time to the cubes would be essentially slower. This makes cubical sets and bitmaps a perfect marriage, which is not possible for other types of sets, for instance sets of simplices and general polyhedra. However, the generic approach to writing the code enables its adaptation to other methods of storing cubical sets and, with some more effort, to simplicial homology. This work is in progress.

9. Experiments

In this section we will compare the performance of the implementations **ASH**, **ASLT**, **AS** and **ASR** in various settings with the performance of the implementations **PP**, **BK** and **AR** described in Section 1.2. Let us emphasize

Table 1
Homology computation times in seconds for two-dimensional torus

Size	**ASLT**	**ASR**	**AS**	**BK**	**ASH**	**AR**	**PP**
80,000	0.2	3.94	9.08	43.48	132	131.31	566.92
115,200	0.28	5.75	14.08	60.27	184.77	226.31	1143.67
156,800	0.36	7.53	20.63	75.95	250.2	361.34	2095.89
204,800	0.45	9.97	25.72	122.44	327.53	547.59	3604.52
259,200	0.58	13.39	33.92	213.8	420.72	785.78	5908.92
320,000	0.69	18.83	43.16	246.41	537.78	1137.06	9197.56
α	0.9	1.1	1.1	1.3	1.0	1.6	2.0

that all these implementations are compatible in the sense that they all constitute part of the Computational Homology Project [34] and may be easily compiled together with the benchmark software in one executable, which makes the comparison straightforward and reliable. To do the experiments, we used a gcc compiler of version 3.4.2 ported for MS Windows XP. The timings presented in this section were obtained on a 3.6 GHz Pentium PC with 2 GB RAM running MS Windows XP. The size of cubical sets in the tables we present is measured as the amount of full elementary cubes in the set.

9.1. Torus

In our first experiment we rescale a two-dimensional cubical torus in directions parallel to its surface at six different scales and compute the homology. The computation times in seconds for various scales and algorithms are gathered in Table 1.

This table shows that the implementations **ASLT**, **ASR** and **AS** significantly outperform, in this case, the other implementations. As one can expect, the winner is **ASLT**, which runs about four orders of magnitude faster than the slowest implementation and more than two orders of magnitude faster than the quickest implementation not based on acyclic subspace construction. The bottom row of Table 1 contains an approximate measure of complexity α of the implementations obtained by finding the best fit of the data to the function $T = cn^{\alpha}$. For all the implementations of the acyclic subspace homology algorithm these numbers are close to one, which indicates that the complexity of these algorithms is close to linear.

The size of the constructed acyclic subset for the four implementations based on Algorithm 2 is about 50% of the original size in the case of **AS** but it exceeds 99% in all the other cases. It is surprising that in the case of **ASR** the reduction is exactly the same as in the case of **ASLT** and **ASH**. The implementations, **ASLT** and **ASH** use a total acyclicity test, but we cannot claim that **ASR** uses a total acyclicity test. Nevertheless, it seems that at least in low dimensions the acyclicity test used by **ASR** might behave in practice as good as a total acyclicity test.

A cubical torus has few removable cubes. There are about 2.5% removable cubes at the lowest rescaling factor and this goes down to 1.25% for the largest rescaling factor. Therefore, one should not expect a substantial gain from preprocessing the algorithms by shaving in this case. Actually, in experiments one even observes an increase in computation time which may go above 400% in the case of **ASR** and above 35% in the case of **ASLT**. This is understandable: when running shaving, every cube is tested for removability and if the cube is not removable, then it contributes to the total computation time only on the sides of expenses. Therefore, the cost is especially visible in algorithms in which the acyclicity test is particularly expensive.

9.2. Bing's house

Our next experiment concerns the cubical Bing's house [41] presented in Fig. 1. Bing's house is a deformation retract of a cube in $\mathbb{R}^3$; therefore, it is acyclic. However, one can show that the acyclic subset of Bing's house constructed by Algorithm 2 cannot be equal to the whole Bing's house. This is related to the fact that the Bing's house is an example of a contractible cubical set which is not collapsible. Therefore, this example constitutes some challenge to Algorithm 2.

Similarly as in the previous experiment, we rescale the Bing's house and compute the homology. The computation times in seconds for various scales and algorithms are gathered in Table 2.

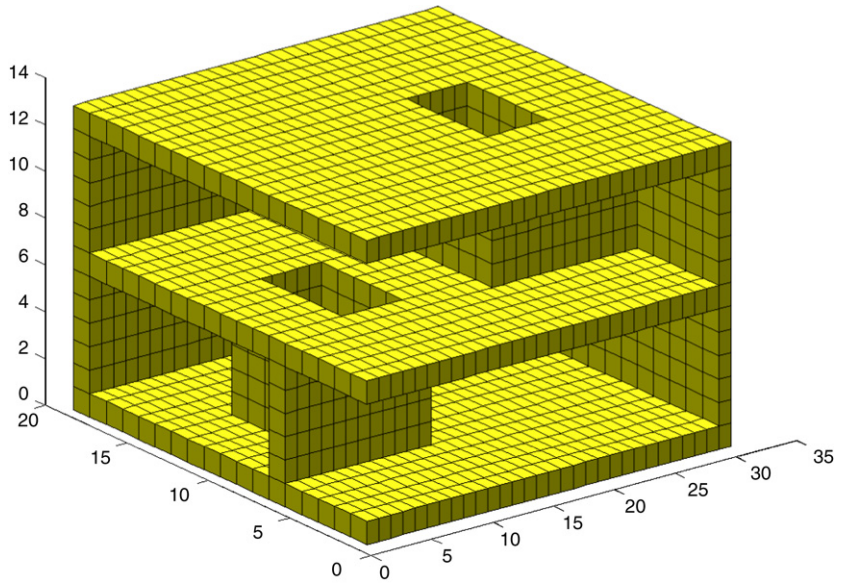

Fig. 1. Cubical Bing's house with two front faces removed.

Table 2
Homology computation times in seconds for Bing's house

Size	**ASLT**	**ASR**	**AS**	**BK**	**ASH**	**AR**	**PP**
74,341	0.16	4.64	11.38	31.53	96.17	41.47	572.39
132,321	0.27	8.34	20.8	57.86	166.98	104.64	1,811.13
206,901	0.41	13.74	33.66	76.31	256.95	212.41	4,180.47
298,081	0.59	20.77	50.88	182.41	370.59	358.31	7,431.42
405,861	0.86	29.56	69.63	221.33	504.7	602.91	12,647.3
530,241	1.11	39.19	90.61	384.99	654.49	–	17,527.7
α	1.0	1.1	1.1	1.3	1.0	1.6	1.8

Despite the challenging character of the Bing's house example, the outcome of this experiment is very similar to the previous one. The graphical comparison of the three best implementations of the acyclic subspace homology algorithm with the best implementation not based on acyclic subspace is presented in Fig. 2. The constructed acyclic subset is again about 50% of the original size in the case of **AS**. In the case of the three other acyclic subspace algorithms this number varies from 97.6% in the case of the smallest rescaling to 99.7% in the case of the largest rescaling. Again, as one may expect, shaving does not speed up the computations in this case but actually it slows them down by factors similar to the case of the torus.

9.3. Klein bottle

Let us turn now our attention to dimension four and consider the Klein bottle. We apply the same procedure of rescaling as in the two preceding examples. The resulting computation times in seconds are gathered in Table 3. The **ASLT** implementation is not available in dimension four, because the lookup tables are too large in this dimension, so we do not run this test. We also skip the **ASH** implementation, because it is very slow. The table shows that also in this case the construction of acyclic subspace substantially speeds up the homology computations.

9.4. Cahn–Hilliard equations

Cahn–Hilliard equation [42] is a phenomenological model used to describe phase separation in binary alloys. The solution of the equation is a function of the time t and location x, which represents the relative concentration difference between the two materials at time t and location x. The change in time of the topology of the set $P(t)$ of

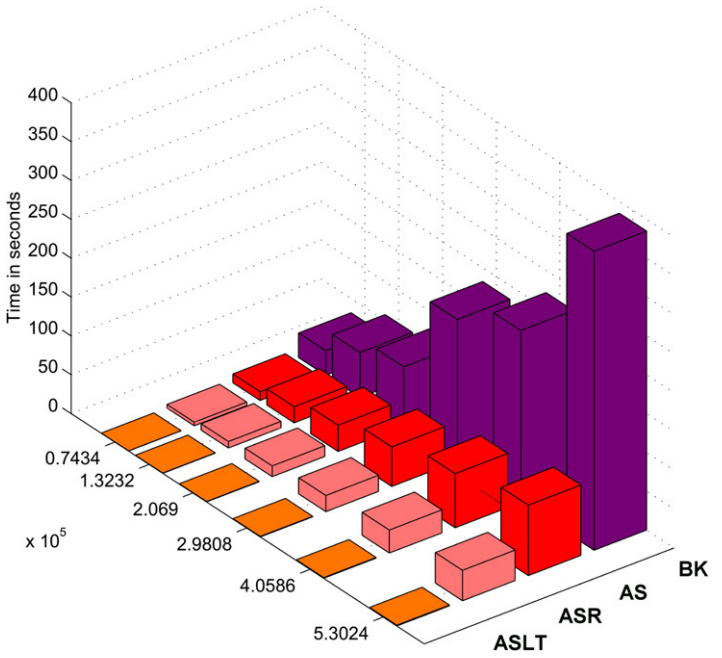

Fig. 2. Graphical comparison of **ASLT**, **ASR**, **AS** and **BK** in the case of Bing's House. The slower implementations are not presented.

Table 3
Homology computation times in seconds for Klein bottle

Size	**ASR**	**AS**	**AR**	**BK**	**PP**
1382	0.94	0.47	1.7	2.83	1.3
12,522	2.89	4.5	20.17	30.52	28.73
34,830	7.28	13.88	67.89	98.36	265.34
68,306	13	28.53	157.64	217.88	1165.16
112,950	20.89	48.66	306.86	373.83	2997.63
168,762	31.48	73.91	–	578.59	6190.05
α	0.7	1.1	1.2	1.1	1.8

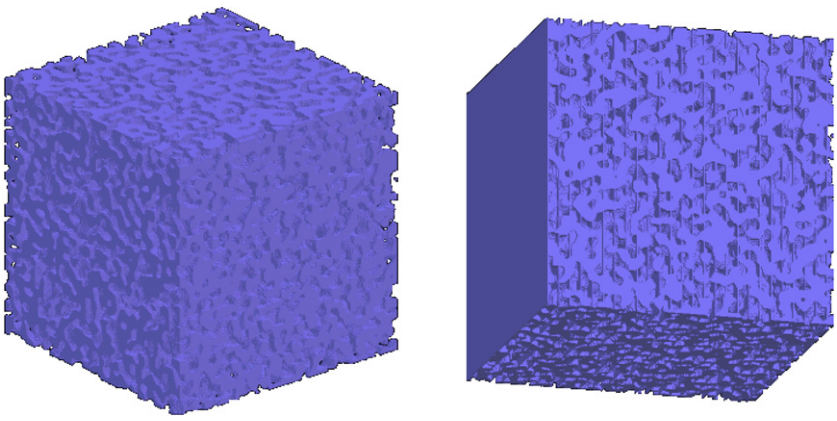

Fig. 3. An unmasked (left) and masked (right) example from Cahn–Hilliard equation.

locations where the function is positive (or of the set $N(t)$ where the function is negative) may be used to identify and distinguish the evolving microstructures described by the Cahn–Hilliard equation [19]. Fig. 3 presents two cubical

Table 4
Homology computation times in seconds for the unmasked example from Cahn–Hilliard equation

Size	**ASLTsh**	**BKsh**	**ASLT**	**BK**
8,392,997	6.55	137.28	195.83	22,952.0

Table 5
Second Betti numbers for the masked example from Cahn–Hilliard equation

Distance	64	16	4
Second Betti number	0	1	130

Table 6
Homology computation times in seconds for the masked examples from Cahn–Hilliard equation

Distance	Size	**ASLTsh**	**BKsh**
64	8,524,217	8.02	196
16	8,917,371	9.55	237.73
4	10,490,025	147.69	1193.31

sets, both on a cubical grid $256 \times 256 \times 256$. The set on the left is a cubical approximation to a sample set $P(t)$. Gameiro [39] found that typically such sets are connected (the 0th Betti number is one), have no voids (the second Betti number is zero) and have many tunnels (the first Betti number is large). Homology computation times for one such set consisting of 8,392,997 three-dimensional cubes are presented in Table 4 for two implementations in two variants: preceded with shaving (**ASLTsh** and **BKsh**) and without shaving (**ASLT** and **BK**). As one can see, shaving dramatically speeds up the computations. Actually, there are 8,302,485 removable cubes in this case (almost 99% of the total number of cubes), so the speeding up should not be surprising. Nevertheless, there are still 90,512 cubes left for further processing. Shaving takes in this case 5.36 s of processor time. Taking into account only the time needed by **ASLTsh** and **BKsh** after shaving, one can again see that **ASLT** is two orders of magnitude faster than **BK**.

As we already mentioned, the simulations together with homology computations indicate the presence of many tunnels and no voids in the sets $P(t)$ and $N(t)$. By inserting a number of equally separated, parallel full planes into these sets (referred to as *masks*) one obtains what we call a *masked set*. An example of such a set is visible on the right-hand side of Fig. 3. The homology of the masked sets may provide some rough measure of the size of the tunnels. The reason is that if the masks are close enough one to the other, then they close the tunnels, which results in the appearance of voids not present in the original sets. Therefore, the second Betti number of the masked sets counts the number of tunnels which close after inserting the mask.

The outcome of an experiment in which mask were inserted respectively in the distance of 64, 16 and 4 voxels are gathered in Table 5. The table shows that the tunnels in general are rather short.

Of more interest to us are the computation times for the masked examples. They are gathered in Table 6. The unshaved implementations are skipped due to long computation times. The interesting thing one can observe is the dramatic increase in the computation times for the cases of masks separated only by 4 voxels. This increase cannot be attributed to the increase in the size of the set. What happens is that in this case shaving reduces the set only to 2,316,567 cubes. This process takes 7.75 s. The acyclic subset of 2,012,701 is constructed in 5.55 s. This leaves still 303,866 cubes for further processing by the relative homology software.

We finish this example with an intriguing diagram. The mask separated by 10 voxels were inserted into the sets $P(t)$ for a sequence of 100 consecutive times t. For every such set the homology was computed. The ratio of the second to the first Betti number is presented in Fig. 4. The ratio may be treated as some measure of the number of long tunnels to all tunnels. Of course, the explanation of the visible oscillations of this measure is beyond the area of the present research. Let us only mention, that the total homology computations needed to produce this graph took less than 2000 s.

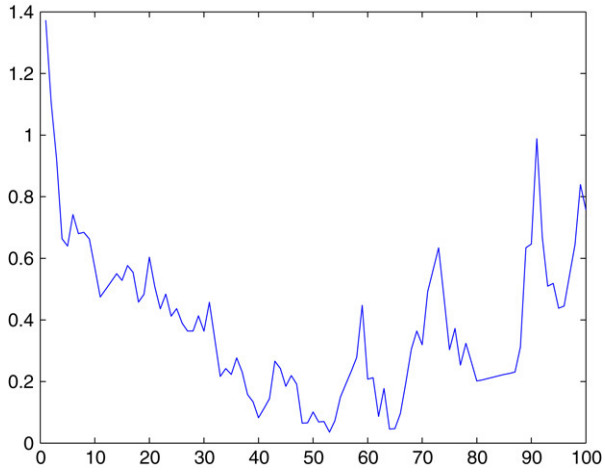

Fig. 4. Ratio β_2/β_1 in masked examples from Cahn–Hilliard equation as the function of time.

Table 7
Homology computation time in seconds of Klein bottle in various sizes for **AR** and **LS**

Size	**AR**	**LS**
380	0.380	0.348
866	0.971	0.929
1,682	1.974	1.957
2,487	3.146	3.349
10,128	15.87	24.95
42,323	87.12	608.3

10. Comparison with other packages

The comparison of various homology algorithms presented in the previous section concern only implementations available from the Computational Homology Project webpage [34]. As we already mentioned, in these cases the tests are reliable, because all these implementations may be compiled together with the benchmark software in one executable. However, other homology algorithms and other homology software are described in the literature.

In this section we present some comparison of the compare homology software available from CHOMP webpage [34] with two other homology algorithms, not included in CHOMP. We presents these comparisons separately, because they are not as reliable as the comparisons presented in the previous section due to difficulties in compiling the software together or differences in the accepted input.

10.1. LinBox library

LinBox project [43] is a project devoted to the exact computational linear algebra. In particular the LinBox library contains software for Smith diagonalization based on recent advances in Smith diagonalization algorithms. The LinBox library is very large, so it is not easy to compile it together with other packages. Recently Urbańska [44] compared **AR** and **ASLT** algorithms with homology algorithms utilizing sparse implementations of Smith diagonalization algorithms available in the LinBox library [43]. The experiments where performed on SGI Altix with 64 Itanium 2 processors. She found that without some form of preliminary reduction the algorithms in LinBox cannot compete with **AS** or even **AR**, at least not for the class of problems, where the complex is large and the homology to be computed is simple.

Table 7 compares **AR** against **LS**, the best performing homology algorithm based on LinBox software for Smith Normal Form combined with a simple form of reduction based on eliminating rows and columns with exactly one nonzero entry. The table shows that the performance of **AR** and **LS** is comparable for small sizes of our four-dimensional representation of Klein bottle, but **AR** is significantly better for large sizes. As pointed out in [44],

Table 8
Comparison of **ASLTsh** and **LS** on an unmasked example from Cahn–Hilliard equation

Size	**ASLTsh**	**LS**
8,393,324	10.94	15,022.9

this is because the reduction algorithm described in [26] uses cascade type eliminations of generators between the dimensions whereas the standard Smith Normal Form algorithm does not benefit from the possible cancellations of generators in different dimensions.

Another test performed by A. Urbańska concerns a direct comparison of **ASLTsh** and **LS** on one of the sets coming from numerical simulations of Cahn–Hillard equations performed by Gameiro [39]. The timings are gathered in Table 8.

10.2. Persistence homology algorithm

The Klein bottle example gives a possibility of a very rough comparison of the acyclic subspace homology algorithm with the persistence homology algorithm [32] by Zomorodian and Carlsson. The implementation of this algorithm is for various field coefficients. The case of $\mathbb{Z}_2$ coefficients is special, because there is no need to store coefficients and the authors have a specialized implementation for this case, which is much faster than the general fields. However, $\mathbb{Z}_2$ coefficients are not good when one is interested in torsion. To detect torsion one needs $\mathbb{Z}$ coefficients or at least $\mathbb{Z}_p$ coefficients for some $p > 2$. Integer coefficients are better, because they guarantee that in every case all torsion is picked up.

Zomorodian and Carlsson present in [32] the timings resulting from the 2.2 GHz Pentium processor homology computations of a simplicial representation of Klein bottle consisting of 12,000 simplices. The timings are 0.01 s for $\mathbb{Z}_2$ coefficients, 0.23 s for $\mathbb{Z}_3$, $\mathbb{Z}_5$ and $\mathbb{Z}_{3203}$ coefficients and 0.5 s for rational coefficients.

Our cubical representation of Klein bottle presented in the first row of Table 3 (for the rescaling factor 1) consists of 1382 four-dimensional cubes. This in contrast to the Zomorodian and Carlsson representation, which is two-dimensional. When all the lower-dimensional faces are counted as in the case of Zomorodian and Carlsson, the number is 49,500. The computation time by **AS** implementation is 0.47 s, but this would be 0.77 s when rescaled to the speed of 2.2 GHz processor and 0.19 s when rescaled to the size of 12,000 simplices. These rough estimates suggest that in the case of Klein bottle the performance of our algorithm may be slightly better or similar to the performance of the persistence homology algorithm except the case of $\mathbb{Z}_2$ coefficients. However, this does not take into account that our representation consists of four-dimensional cubes, whereas the Zomorodian and Carlsson representation is built of two-dimensional simplices.

The true comparison of acyclic subspace homology algorithm with the persistence homology algorithm of Zomorodian and Carlsson will be possible only after the acyclic subspace algorithm in various versions is implemented for simplicial complexes and the computations are compared on the same hardware for various rescalings of Klein bottle and other sets.

11. Conclusions

In the present paper we introduced a new homology algorithm based on the construction of an acyclic subspace. We proved that the complexity of the construction of the acyclic subspace is linear. We considered four tests for the acyclicity, leading to four different variants of the acyclic subspace homology algorithm. Then we presented several numerical experiments with the implementation of the four variants for cubical homology. The tests clearly indicate that the implementation of the version based on lookup tables for dimensions two and three significantly outperform other available software for cubical homology. This applies both to purely artificial tests based on rescaled, simple topological spaces and to a test based on data gathered from numerical investigation of differential equations. In dimensions higher than three the superiority is not so strong. This is because the number of neighbors of a cube grows exponentially with dimension. In particular, in dimension higher than three we cannot use lookup tables. A method to circumvent this problem will be presented in [36].

On the theoretical side it would be nice to understand how deep the reduction based on acyclic subspace construction may be. The numerical tests, even in the case of Bing's house, indicate that the reduction is very

substantial. But we are not able to exclude the existence of spaces with complicated simple homotopy type [45] (for instance along the lines of Bing's house), for which it is not possible to construct a large acyclic subspace despite the fact that their homology is simple. However, let us mention the following conjecture concerning Algorithm 2.

Conjecture 17 (*[46]*). *Let $\mathcal{X}_n$ be a sequence of cubical families such that* $\operatorname{card} \mathcal{X}_n \to \infty$ *and for any two $n, m \in \mathbb{N}$, the sets $|\mathcal{X}_n|$ and $|\mathcal{X}_m|$ are homeomorphic. Then*

$$\lim_{n\to\infty} \frac{\operatorname{card} \texttt{AcyclicSubspace}(\mathcal{X}_n)}{\operatorname{card} \mathcal{X}_n} = 1.$$

If the conjecture is true, then together with Theorem 3 it would imply that the homology of a cubical set of a fixed topology type may be computed in linear time. An analogous conjecture may be formulated for simplicial complexes.

Apart from trying to understand the theoretical aspects of the construction of acyclic subspace, there are several directions in which the present research may be continued. The first thing to do is to adapt the available implementation to simplicial homology and compare it with the available software for simplicial homology. This work is in progress. Also the adaptation of the method to the computation of homology of inclusions [47] is in progress. Definitely it is worth to investigate some other methods of testing for acyclicity, better than the simple intersection method described in this paper, but still computationally inexpensive. This is left for future investigation.

Acknowledgments

We express our thanks to M. Gameiro for the permission to use the results of his numerical simulations of the Cahn–Hilliard equation in the experiments presented in this paper. Gameiro's simulations were performed with the support of a DARPA grant.

The first and the third author are partially supported by KBN, Grant N201 037 31/3151. The second author is partially supported by the DARPA TDA project and the Department of Energy grant no. 97891.

References

[1] B.R. Donald, D.R. Chang, On the complexity of computing the homology type of a triangulation, in: Proc. 32nd Ann. IEEE Sympos. Found. Comput. Sci. 1991, pp. 650–661.
[2] R.E. Moore, Interval Analysis, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1966.
[3] K. Mischaikow, M. Mrozek, Chaos in Lorenz equations: A computer assisted proof, American Mathematical Society. Bulletin. New Series 33 (1995) 66–72.
[4] K. Mischaikow, M. Mrozek, Chaos in the Lorenz equations: A computer assisted proof, part II: Details, Mathematics of Computation 67 (1998) 1023–1046.
[5] P. Pilarczyk, Homology computation — software and examples. http://www.pawelpilarczyk.com/homology.php, 1999.
[6] W. Kalies, Chom — A cubical homology program. http://www.math.fau.edu/kalies/chom.html, 1999.
[7] A. Szymczak, A combinatorial procedure for finding isolating neighborhoods and index pairs, Proceedings of the Royal Society of Edinburgh, Series A 127A (1997) 1075–1088.
[8] A. Szymczak, Index pairs: From dynamics to combinatorics and back, Ph.D. Thesis, Georgia Inst. Tech., Atlanta, 1999.
[9] M. Dellnitz, O. Junge, The algorithms behind GAIO — set oriented numerical methods for dynamical systems, in: Ergodic Theory, Analysis, and Efficient Simulation of Dynamical Systems, Springer, Berlin, 2001, pp. 145–174, 805–807.
[10] M. Mrozek, Index pairs algorithms, Foundations of Computational Mathematics 6 (2006) 457–493.
[11] P. Pilarczyk, Computer assisted method for proving existence of periodic orbits, TMNA 13 (1999) 365–377.
[12] S. Day, Towards a rigorous numerical study of the Kot–Schaffer model, Dynamic Systems and Applications 12 (2003) 87–98.
[13] S. Day, O. Junge, K. Mischaikow, A rigorous numerical method for the global analysis of infinite dimensional discrete dynamical systems, SIAM Dynamical Systems 3 (2004) 117–160.
[14] Z. Arai, K. Mischaikow, Rigorous computations of homoclinic tangencies, SIAM Journal on Applied Dynamical Systems 5 (2006) 280–292.
[15] M. Allili, K. Mischaikow, A. Tannenbaum, Cubical homology and the topological classification of 2D and 3D imagery, IEEE International Conference on Image Processing 2 (2001) 173–176.
[16] M. Niethammer, A.N. Stein, W.D. Kalies, P. Pilarczyk, K. Mischaikow, A. Tannenbaum, Analysis of blood vessel topology by cubical homology, in: Proceedings of International Conference on Imagine Processing, vol. 2, 2002, pp. 969–972.
[17] W. Kalies, M. Niethammer, K. Mischaikow, A. Tannenbaum, On the detection of simple points in higher dimensions using cubical homology, IEEE Transactions on Image Processing 15 (2006) 2462–2469.
[18] M. Żelawski, Pattern recognition based on homology theory, Machine Graphic and Vision 14 (2005) 309–324.
[19] M. Gameiro, K. Mischaikow, Th. Wanner, Evolution of pattern complexity in the Cahn–Hilliard theory of phase separation, Acta Materialia 53 (2005) 693–704.

[20] M. Gameiro, W. Kalies, K. Mischaikow, Topological characterization of spatial-temporal chaos, Physical Review E 70 (2004) Article 035203 (Rapid communication).
[21] V. de Silva, R. Ghrist, Coordinate-free coverage in sensor networks with controlled boundaries via homology, International Journal of Robotics Research 25 (2006) 1205–1222.
[22] T. Teramoto, Morphological characterization of diblock copolymer problem and topological computation, Conference presentation available at: http://chomp.rutgers.edu/workshop/.
[23] J.R. Munkres, Elements of Algebraic Topology, Addison-Wesley, 1984.
[24] A. Storjohann, Near optimal algorithms for computing Smith normal form of integer matrices, in: Proceedings of the 1996 International Symposium on Symbolic and Algebraic Computation, ISAAC 1996, 1996, pp. 267–274.
[25] J. Friedman, Computing Betti numbers via combinatorial Laplacians, in: Proc. 28th Ann. ACM Sympos. Theory Comput., 1996, pp. 386–391.
[26] T. Kaczynski, M. Mrozek, M. Ślusarek, Homology computation by reduction of chain complexes, Computers and Mathematics with Applications 35 (1998) 59–70.
[27] W. Kalies, K. Mischaikow, G. Watson, Cubical Approximation and Computation of Homology, in: Conley Index Theory, vol. 47, Banach Center Publications, 1999, pp. 115–131.
[28] S. Basu, On bounding the Betti numbers and computing the Euler characteristic of semi-algebraic sets, Discrete and Computational Geometry 22 (1999) 1–18.
[29] H. Edelsbrunner, D. Letscher, A. Zomorodian, Topological persistence and simplification, Discrete and Computational Geometry 28 (2002) 511–533.
[30] J.-G. Dumas, F. Heckenbach, D. Saunders, V. Velker, Computing simplicial homology based on efficient Smith normal form algorithms, in: Algebra, Geometry and Software Systems, 2003, pp. 177–207.
[31] T. Kaczynski, K. Mischaikow, M. Mrozek, Computational homology, in: Applied Mathematical Sciences, vol. 157, Springer-Verlag, New York, 2004.
[32] A. Zomorodian, G. Carlsson, Computing persistent homology, Discrete and Computational Geometry 33 (2005) 249–274.
[33] K. Mischaikow, M. Mrozek, P. Pilarczyk, Graph approach to the computation of the homology of continuous maps, Foundations of Computational Mathematics 5 (2005) 199–229.
[34] Computational homology project. http://chomp.rutgers.edu/.
[35] M. Mrozek, Homology software. http://www.ii.uj.edu.pl/~mrozek/software/homology.html, 2006.
[36] M. Mrozek, B. Batko, The coreduction homology algorithm (submitted for publication).
[37] J. Blass, W. Holsztyński, Cubical polyhedra and homotopy, I, II, III, IV, V, Atti della Accademia Nazionale dei Lincei Rendiconti. Classe di Scienze Fisiche, Matematiche e Naturali 50 (2) (1971) 131–138; 50(6) (1971) 703–708; 53(8) (1972) 275–279; 53(8) (1972) 402–409; 54 (1973) 416–425.
[38] M. Gameiro, V. Nanda, Modifications to Kalies' homology software, Personal communication.
[39] M. Gameiro, Numerical simulations of the 3D Cahn–Hilliard equation, Personal communication.
[40] Computer assisted proofs in dynamics. http://capd.wsb-nlu.edu.pl.
[41] R.H. Bing, Some aspects of the topology of 3-manifolds related to the Poincaré conjecture, in: T.L. Saaty (Ed.), Lectures on Modern Mathematics II, Wiley, 1964, pp. 93–128.
[42] J.W. Cahn, J.E. Hilliard, Free energy of a nonuniform system. I. Interfacial free energy, Journal of Chemical Physics 28 (1958) 258–267.
[43] Project LinBox: Exact computational linear algebra. http://www.linalg.org.
[44] A. Urbańska, Smith normal form algorithms for sparse matrices with applications to homology computations, M.Sc. Thesis, Jagiellonian University, Kraków, 2007 (in Polish, with English abstract).
[45] M. Cohen, A course in simple-homotopy theory, in: Graduate Texts in Mathematics, vol. 10, Springer-Verlag, New York, Berlin, 1973.
[46] T. Kaczynski, M. Mrozek, R. Srzednicki, Personal communication.
[47] N. Żelazna, Computing homology of inclusions via acyclic subspace construction, Schedae Informaticae (in press).